5COSC005W MOBILE APPLICATION DEVELOPMENT
Lecture 5: Working with Databases

Torin Wirasingha

SQLite

SQLite is a tiny yet powerful database engine.

Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

SQLite

SQLite is a tiny yet powerful database engine.
Besides Android, it can be found in:

- Apple iPhone
- Symbian phones
- Mozilla Firefox
- Skype
- PHP
- Adobe AIR
- Mac OS X
- Solaris
- many others...

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

Android stores it in the `/data/data/packagename/databases` directory.

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

Android stores it in the `/data/data/packagename/databases` directory.

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

Android stores it in the `/data/data/packagename/databases` directory.

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

Android stores it in the `/data/data/packagename/databases` directory.

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

Android stores it in the `/data/data/packagename/databases` directory.

Advantages of SQLite

- It's free.
- It's small. The current version is about 150KB.
- It requires no setup or administration. There is no server, no config file, and no need for a database administrator.

A SQLite database is just a file. You can take that file, move it around, and even copy it to another system.

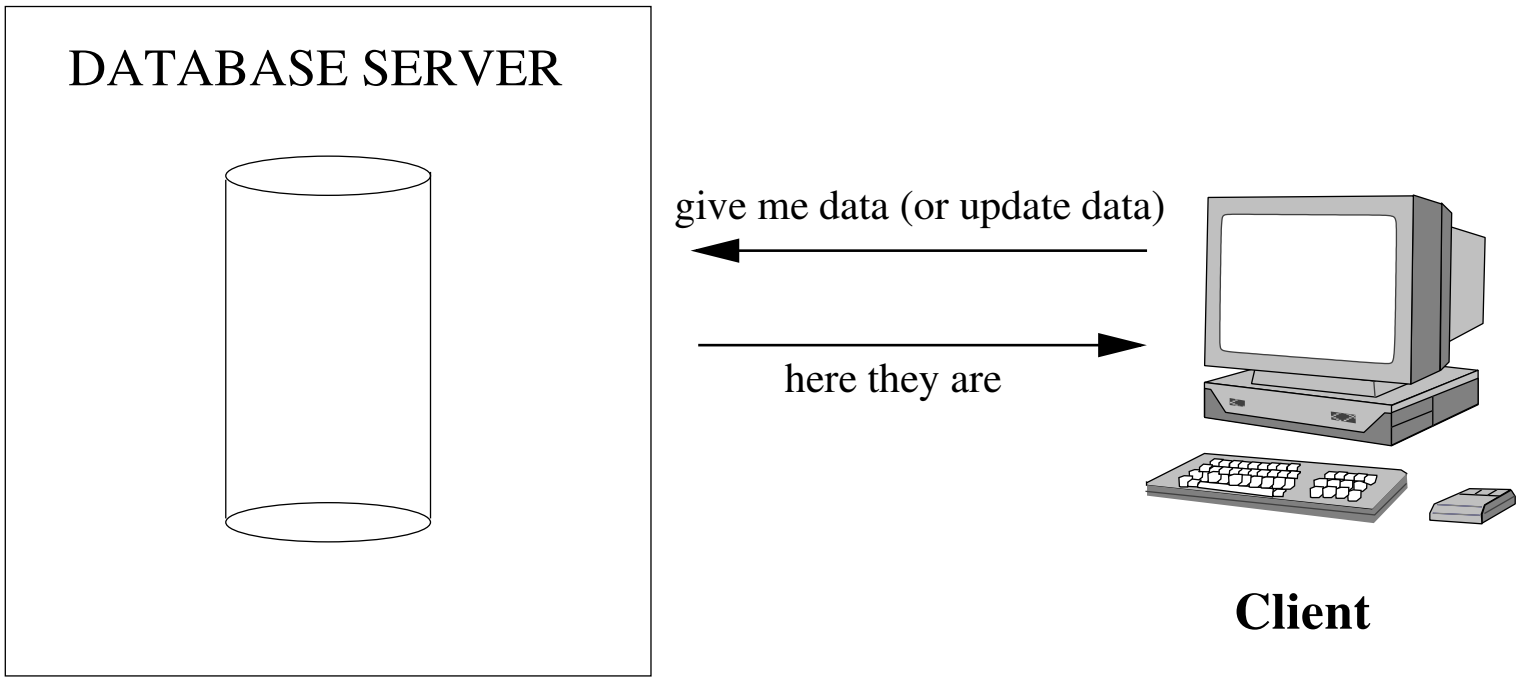
Android stores it in the `/data/data/packagename/databases` directory.

What is a Database Server

Just another server which receives requests from clients requiring access to data in a database (this could be read or write).

What is a Database Server

Just another server which receives requests from clients requiring access to data in a database (this could be read or write).



Relational Databases

Everything organised into tables.

Name	Age	Position	Salary
John Smith	35	Manager	40000
Robert Barclay	28	Developer	30000
George Deval	25	Administrator	32000
Tom Bubble	38	Head of Sales	45000

Relational Databases

Everything organised into tables.

Name	**Age**	**Position**	**Salary**
John Smith	35	Manager	40000
Robert Barclay	28	Developer	30000
George Deval	25	Administrator	32000
Tom Bubble	38	Head of Sales	45000

Accessing Databases

SQL (Structured Query Language) is used.

The main variations are:

- Transact SQL (T-SQL). Used by Microsoft SQL Server and Sybase. The two have very few differences.
- PL-SQL. Used in Oracle.
- ANSI SQL. Parts of it adopted by commercial and public domain products.

Accessing Databases

SQL (Structured Query Language) is used.

The main variations are:

- Transact SQL (T-SQL). Used by Microsoft SQL Server and Sybase. The two have very few differences.
- PL-SQL. Used in Oracle.
- ANSI SQL. Parts of it adopted by commercial and public domain products.

Accessing Databases

SQL (Structured Query Language) is used.

The main variations are:

- Transact SQL (T-SQL). Used by Microsoft SQL Server and Sybase. The two have very few differences.
- PL-SQL. Used in Oracle.
- ANSI SQL. Parts of it adopted by commercial and public domain products.

Accessing Databases

SQL (Structured Query Language) is used.

The main variations are:

- Transact SQL (T-SQL). Used by Microsoft SQL Server and Sybase. The two have very few differences.
- PL-SQL. Used in Oracle.
- ANSI SQL. Parts of it adopted by commercial and public domain products.

SQL Datatypes

SQL Datatype	**Corresponding Java type**
CHAR(n)	String
VARCHAR(n)	String
INTEGER or INT	int
DOUBLE	double
DATE	java.sql.Date
TIMESTAMP	java.sql.Timestamp
DECIMAL, NUMERIC	java.math.BigDecimal

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

SQL Statements

Four main categories:

- CREATE and INSERT (create a table, put values into it)
- SELECT (query the database about data matching certain criteria)
- UPDATE (to change the values in existing rows)
- DELETE and DROP (to delete specific rows or tables).

The CREATE Statement

Syntax:

```
CREATE TABLE tablename(
    colName  dataType
)
```

Example:

```
CREATE TABLE Person (
   name VARCHAR(100),
   age  INTEGER,
   address VARCHAR(100))
```

The CREATE Statement

Syntax:

```
CREATE TABLE tablename(
    colName  dataType
)
```

Example:

```
CREATE TABLE Person (
   name VARCHAR(100),
   age  INTEGER,
   address VARCHAR(100))
```

The INSERT Statement

Syntax:

```
INSERT INTO tablename
    (colName1, colName2, colName3 ...)
VALUES
    (value1, value2, value3, ...)
```

Example:

```
INSERT INTO Person (name, age, address)
VALUES ('John Smith', 26, 'London'),
       ('Tom Bubble', 34, 'New York')
```

The INSERT Statement

Syntax:

```
INSERT INTO tablename
    (colName1, colName2, colName3 ...)
VALUES
    (value1, value2, value3, ...)
```

Example:

```
INSERT INTO Person (name, age, address)
VALUES ('John Smith', 26, 'London'),
       ('Tom Bubble', 34, 'New York')
```

The SELECT Statement

Syntax:

```
SELECT
   Name1, Name2, Name3 ...
FROM tablename1, tablename2, ...
WHERE
   conditions
ORDER BY colNames
```

Example:

```
SELECT Person.name, Person.address,
       ListensTo.music_group_name
FROM Person, ListensTo
WHERE ListensTo.music-group_name IN ('Beatles',
                                     'Popstars')
AND Person.name = ListensTo.person_name
AND Person.address = 'London'
```

The SELECT Statement

Syntax:

```
SELECT
   Name1, Name2, Name3 ...
FROM tablename1, tablename2, ...
WHERE
   conditions
ORDER BY colNames
```

Example:

```
SELECT Person.name, Person.address,
       ListensTo.music_group_name
FROM Person, ListensTo
WHERE ListensTo.music-group_name IN ('Beatles',
                                     'Popstars')
AND Person.name = ListensTo.person_name
AND Person.address = 'London'
```

The UPDATE Statement

Syntax:

```
UPDATE tablename
  SET colName1=value1, colName2=value2 ...
  WHERE colNamei someOperator valuei
```

Example:

```
UPDATE Person
  SET age = 25, address='Manchester'
  WHERE name = 'John Smith'
```

The UPDATE Statement

Syntax:

```
UPDATE tablename
  SET colName1=value1, colName2=value2 ...
  WHERE colNamei someOperator valuei
```

Example:

```
UPDATE Person
  SET age = 25, address='Manchester'
  WHERE name = 'John Smith'
```

The DELETE and DROP Statements

Syntax:

```
DELETE FROM tablename
    WHERE colNamei someoperator valuei
```

Example:

```
DELETE FROM Person
    WHERE name = 'John Smith'
```

The rows corresponding to `John Smith` are deleted.

- To delete a whole table (not only the contents but the table itself) use the DROP statement. (after that the table needs to be created again).

Example:

```
DROP TABLE Person
```

The DELETE and DROP Statements

Syntax:

```
DELETE FROM tablename
    WHERE colNamei someoperator valuei
```

Example:

```
DELETE FROM Person
    WHERE name = 'John Smith'
```

The rows corresponding to `John Smith` are deleted.

- To delete a whole table (not only the contents but the table itself) use the DROP statement. (after that the table needs to be created again).

Example:

```
DROP TABLE Person
```

The DELETE and DROP Statements

Syntax:

```
DELETE FROM tablename
    WHERE colNamei someoperator valuei
```

Example:

```
DELETE FROM Person
    WHERE name = 'John Smith'
```

The rows corresponding to `John Smith` are deleted.

- To delete a whole table (not only the contents but the table itself) use the DROP statement. (after that the table needs to be created again).

Example:

```
DROP TABLE Person
```

The DELETE and DROP Statements

Syntax:

```
DELETE FROM tablename
    WHERE colNamei someoperator valuei
```

Example:

```
DELETE FROM Person
    WHERE name = 'John Smith'
```

The rows corresponding to `John Smith` are deleted.

- To delete a whole table (not only the contents but the table itself) use the DROP statement. (after that the table needs to be created again).

Example:

```
DROP TABLE Person
```

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

An example of Creating a Table in SQLite

```
create table mytable (
    _id integer primary key autoincrement,
    name text,
    phone text );
```

- One of the columns is designated as the **PRIMARY KEY**, a number that uniquely identifies the row.
- **AUTOINCREMENT** means that the database will add 1 to the key for every record to make sure it's unique.
- By convention, the first column is always called **_id**.
- Unlike most databases, in SQLite the column types are just hints. If you try to store a string in an integer column, or vice versa, it will just work with no complaints.

Building the SQLite Database

1. Check if the database exists
   1. If it does not, create it, create the tables and populate them with initial data
   2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

Building the SQLite Database

1. Check if the database exists
    1. If it does not, create it, create the tables and populate them with initial data
    2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

Building the SQLite Database

1. Check if the database exists
   1. If it does not, create it, create the tables and populate them with initial data
   2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

Building the SQLite Database

1. Check if the database exists
   1. If it does not, create it, create the tables and populate them with initial data
   2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

Building the SQLite Database

1. Check if the database exists
   1. If it does not, create it, create the tables and populate them with initial data
   2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

Building the SQLite Database

1. Check if the database exists
    1. If it does not, create it, create the tables and populate them with initial data
    2. If it does, open it, check what version it is.
2. If the database is an old version, upgrade it to a newer version.

The above steps are facilitated by using the `SQLiteOpenHelper` class.

The SQLiteOpenHelper Class

1. **SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)**: create an object of a subclass of the helper class.
2. Call **getWritableDatabase()** or **getReadableDatabase()** to create or open a database.
3. **onCreate (SQLiteDatabase db)**: Called when the database is created for the first time.
4. **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**: Called when the database needs to be upgraded (new version).

The SQLiteOpenHelper Class

1. **SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)**: create an object of a subclass of the helper class.
2. Call **getWritableDatabase()** or **getReadableDatabase()** to create or open a database.
3. **onCreate (SQLiteDatabase db)**: Called when the database is created for the first time.
4. **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**: Called when the database needs to be upgraded (new version).

The SQLiteOpenHelper Class

1. **SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)**: create an object of a subclass of the helper class.
2. Call **getWritableDatabase()** or **getReadableDatabase()** to create or open a database.
3. **onCreate (SQLiteDatabase db)**: Called when the database is created for the first time.
4. **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**: Called when the database needs to be upgraded (new version).

The SQLiteOpenHelper Class

1. **SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)**: create an object of a subclass of the helper class.
2. Call **getWritableDatabase()** or **getReadableDatabase()** to create or open a database.
3. **onCreate (SQLiteDatabase db)**: Called when the database is created for the first time.
4. **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**: Called when the database needs to be upgraded (new version).

A Hello Database Example

The main activity corresponds to class `MainActivity.java`.

An interface `Constants.java` defining some constants:

```
package org.example.events;
import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
    public static final String TABLE_NAME = "events" ;
    // Columns in the Events database
    public static final String TIME = "time" ;
    public static final String TITLE = "title" ;
}
```

A Hello Database Example

The main activity corresponds to class `MainActivity.java`.
An interface `Constants.java` defining some constants:

```
package org.example.events;
import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
    public static final String TABLE_NAME = "events" ;
    // Columns in the Events database
    public static final String TIME = "time" ;
    public static final String TITLE = "title" ;
}
```

A Hello Database Example

The main activity corresponds to class `MainActivity.java`.
An interface `Constants.java` defining some constants:

```
package org.example.events;
import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
    public static final String TABLE_NAME = "events" ;
    // Columns in the Events database
    public static final String TIME = "time" ;
    public static final String TITLE = "title" ;
}
```

A Hello Database Example (cont'd)

Class EventsData:

```
package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class EventsData extends SQLiteOpenHelper {
   private static final String DATABASE_NAME =
                                        "events.db";
   private static final int DATABASE_VERSION = 1;
```

A Hello Database Example (cont'd)

Class EventsData:

```
package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class EventsData extends SQLiteOpenHelper {
    private static final String DATABASE_NAME =
                                        "events.db";
    private static final int DATABASE_VERSION = 1;
```

A Hello Database Example (cont'd)

```
    /* Create a helper object for the Events database */
    public EventsData(Context ctx) {
      super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE " + TABLE_NAME + " ("
               + _ID
               + " INTEGER PRIMARY KEY AUTOINCREMENT, "
               + TIME + " INTEGER,"
               + TITLE + " TEXT NOT NULL);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
                          int oldVersion,
                          int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
       onCreate(db);
    }
}
```

A Hello Database Example (cont'd)

```
    /* Create a helper object for the Events database */
    public EventsData(Context ctx) {
      super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE " + TABLE_NAME + " ("
              + _ID
              + " INTEGER PRIMARY KEY AUTOINCREMENT, "
              + TIME + " INTEGER,"
              + TITLE + " TEXT NOT NULL);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
                          int oldVersion,
                          int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
       onCreate(db);
    }
}
```

A Hello Database Example (cont'd)

```
    /* Create a helper object for the Events database */
    public EventsData(Context ctx) {
      super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE " + TABLE_NAME + " ("
              + _ID
              + " INTEGER PRIMARY KEY AUTOINCREMENT, "
              + TIME + " INTEGER,"
              + TITLE + " TEXT NOT NULL);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db,
                          int oldVersion,
                          int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
       onCreate(db);
    }
}
```

A Hello Database Example (cont'd)

The main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
    android:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</ScrollView>
```

A Hello Database Example (cont'd)

The `main.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
    android:id="@+id/text"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</ScrollView>
```

A Hello Database Example (cont'd)

The MainActivity class:

```
package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.TextView;
```

A Hello Database Example (cont'd)

The MainActivity class:

```
package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.TextView;
```

A Hello Database Example (cont'd)

```
public class MainActivity extends Activity {
    private static String[] FROM = { _ID, TIME, TITLE,};
    private static String ORDER_BY = TIME + " DESC";
    private EventsData events;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        events = new EventsData(this);
        try {
            addEvent("Hello, Android!");
            Cursor cursor = getEvents();
            showEvents(cursor);
        } finally {
            events.close();
        }
    }
}
```

A Hello Database Example (cont'd)

```
private void addEvent(String string) {
    /* Insert a new record into the Events data
       source. You would do something similar
       for delete and update. */
    SQLiteDatabase db = events.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
    db.insertOrThrow(TABLE_NAME, null, values);
}

private Cursor getEvents() {
    /* Perform a managed query. The Activity will
       handle closing and re-querying the cursor
       when needed. */
    SQLiteDatabase db = events.getReadableDatabase();
    Cursor cursor = db.query(TABLE_NAME, FROM, null,
                             null, null, null,
                             ORDER_BY);
    return cursor;
}
```

A Hello Database Example (cont'd)

```
private void addEvent(String string) {
    /* Insert a new record into the Events data
       source. You would do something similar
       for delete and update. */
    SQLiteDatabase db = events.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
    db.insertOrThrow(TABLE_NAME, null, values);
}

private Cursor getEvents() {
    /* Perform a managed query. The Activity will
       handle closing and re-querying the cursor
       when needed. */
    SQLiteDatabase db = events.getReadableDatabase();
    Cursor cursor = db.query(TABLE_NAME, FROM, null,
                             null, null, null,
                             ORDER_BY);
    return cursor;
}
```

A Hello Database Example (cont'd)

The showEvents method:

```
private void showEvents(Cursor cursor) {
    // Stuff them all into a big string
    StringBuilder builder = new StringBuilder(
            "Saved events:\n");
    while (cursor.moveToNext()) {
        /* Could use getColumnIndexOrThrow() to
           get indexes */
        long id = cursor.getLong(0);
        long time = cursor.getLong(1);
        String title = cursor.getString(2);
        builder.append(id).append(": ");
        builder.append(time).append(": ");
        builder.append(title).append("\n");
    }
    cursor.close();

    // Display on the screen
    TextView text = (TextView) findViewById(
                                    R.id.text);
    text.setText(builder);
}
```

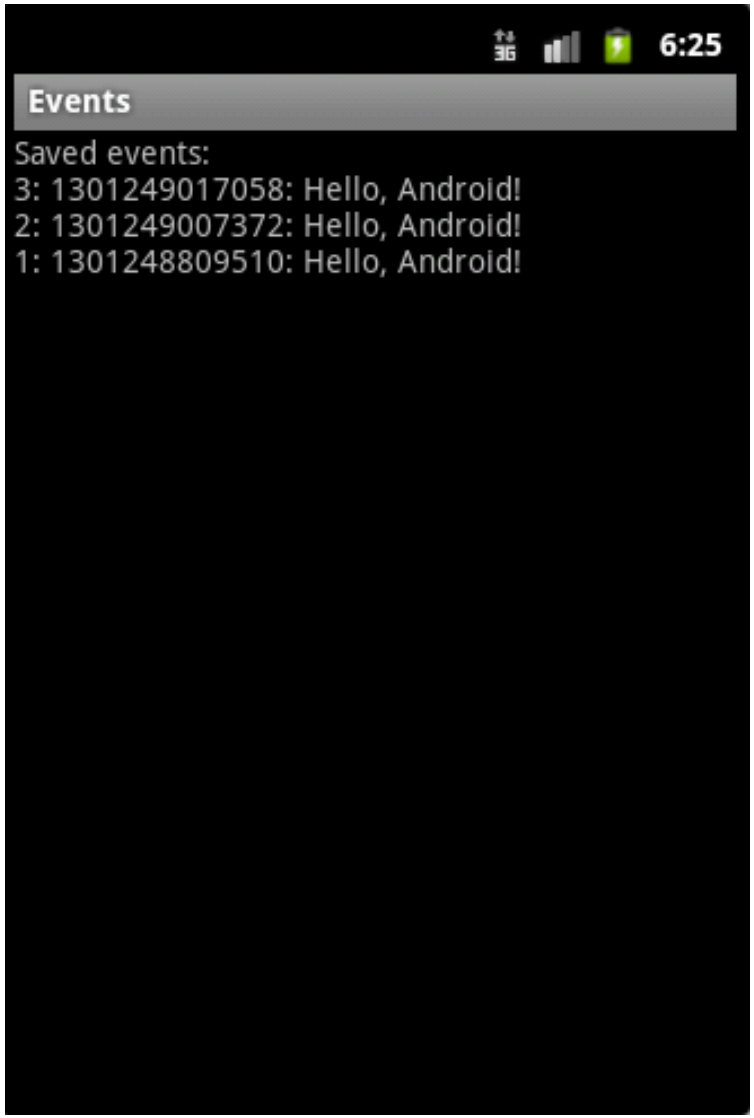
A Hello Database Example (cont'd)

The showEvents method:

```
private void showEvents(Cursor cursor) {
    // Stuff them all into a big string
    StringBuilder builder = new StringBuilder(
            "Saved events:\n");
    while (cursor.moveToNext()) {
        /* Could use getColumnIndexOrThrow() to
           get indexes */
        long id = cursor.getLong(0);
        long time = cursor.getLong(1);
        String title = cursor.getString(2);
        builder.append(id).append(": ");
        builder.append(time).append(": ");
        builder.append(title).append("\n");
    }
    cursor.close();

    // Display on the screen
    TextView text = (TextView) findViewById(
                                        R.id.text);
    text.setText(builder);
}
```

A Hello Database Example (cont'd)



The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

```
adb shell
```

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events

sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```

The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

```
adb shell
```

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events

sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```

The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

```
adb shell
```

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events

sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```

The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

`adb shell`

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events

sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```

The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

```
adb shell
```

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events
```

```
sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```

The sqlite3 utility

You can examine and manipulate the files representing the databases in an emulator (or a rooted device):

`adb shell`

At the shell prompt:

```
sqlite3 /data/data/org.db.databaseexample/databases/events.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

(`org.db.databaseexample` should be the package name of your application).

```
sqlite> .tables
android_metadata  events
```

```
sqlite> select * from events;
1|1488209719643|Hello, Android!
2|1488209728651|Hello, Android!
3|1488209733013|Hello, Android!
4|1488209737209|Hello, Android!
5|1488209740765|Hello, Android!
```